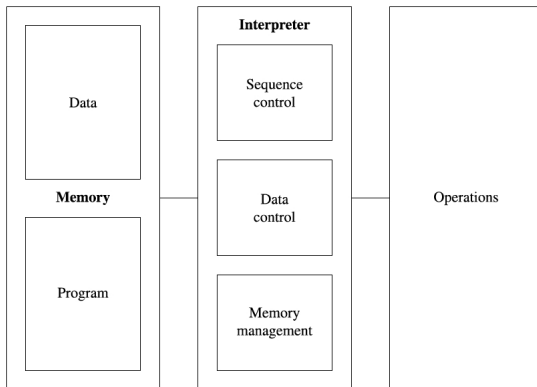


Sommario: 20 aprile, 2021

- Raccomandazioni e Preliminari sul Materiale Prodotto.
- AM21: Architettura e Ruolo dei Componenti.
- AM21: Il Modello Astratto - Stato.
- AM21: Il Modello Astratto - Operazioni sullo Stato ed i suoi Componenti.
- Implementazione dello Stato.
- Attività di Oggi: Esercizi

- Caricare il file Small21, distribuito per la sessione, sulla propria area di sviluppo (terminal, dir. di file system, documents etc).
Verifica: Caricarlo sul REPL OCaml e Controllare Esecuzione Tests Precedenti.
- Copiarlo in Small21XXX, con l'indice corretto.
- Seguire la Presentazione delle Attività della Sessione.
- Svogere le Attività modificando il file Small21 distribuito.

AM21: Architettura e Ruolo dei Componenti



- **Program.** Ogni AST di Small21. Esempio: il programma "uno" (laboratorio1.b)
- **Data.** Modello per soli Atomici int, bool (ed eventuali, char, pointer, modificabili e non).
 - Strutturati array (records, list, etc. solo emulati via Data Control)
- **Memory Management.** Allocazione(/Dealloca) memoria per tutte le richieste.
 - Statica (allocata dinamicamente - dichiarazioni in Seq/Data Control)
 - Stack (Activation Records - Blocks in Seq/Data Control)
 - Heap (solo emulata e solo da estensioni di Small21 - costruito new e simili)
- **Data Control.** Gestisce Frame e interpreta i binding per accedere ai dati.
- **Sequence Control.** Gestisce Activation Frame e Controlla la AM

AM21: Il Modello Astratto - Lo Stato

- **Frame** per identificatori Locali di Entità Denotabili (costanti, variabili, label, array, (nuovi tipi), procedure/funzioni,...)
 - Indicato da una sequenza finita di **bindings** separati da "," racchiusa in parentesi quadre:

$$[Ide_1/Den_1, \dots, Ide_k/Den_k]$$

- **Stack di AR** per Blocchi.
 - Ogni Activation Record è indicato da una quintupla di componenti separati da "," racchiusa in parentesi grafe:

$$\{\text{head, chain, frame, con, val}\}$$

- **Memoria Statica** per Allocare Variabili ed, al momento, i Componenti modificabili di array dei tipi interi e bool
 - Indicata da sequenza finita di **words** separate da "," racchiusa in parentesi quadre:

$$[loc_1 \leftarrow Mv_1, \dots, loc_k \leftarrow Mv_k]$$

- **Stato** è rappresentato una coppia (Δ, μ) , indicante Stack e Memoria.
 - Lo Stato è indicato con la variabile σ (anche con pedici)

● Relazione tra Stato, Semantica ed Operazioni.

- Lo Stato esprime tutte e sole le configurazioni della AM
- Computazione di un programma Small21 P a partire da uno stato S e la Sequenza di Stati che la Semantica associa P nello stato iniziale S.
- La Semantica è espressa nelle sole operazioni di AM

● Funzioni Semantiche e Sistema di Inferenza associato.

- **dclSem** : $Dcl \rightarrow State \rightarrow Type * State$
 $(\forall d, \sigma) \text{ dclSem}(d, \sigma) = (t, \sigma') \text{ iff } \langle d, \sigma \rangle \rightarrow (t, \sigma') \in \mathbf{Sem}_{DCL}$
- **expSem** : $exp \rightarrow State \rightarrow Type * Val * State$
 $(\forall e, \sigma) \text{ expSem}(e, \sigma) = (t, v, \sigma') \text{ iff } \langle e, \sigma \rangle \rightarrow [t, v, \sigma'] \in \mathbf{Sem}_{EXP}$
- **dexpSem** : $dexp \rightarrow State \rightarrow Type * DVal * State$
 $(\forall e_d, \sigma) \text{ expSem}(e_d, \sigma) = (t_d, v_d, \sigma') \text{ iff } \langle e_d, \sigma \rangle \rightarrow [t_d, v_d, \sigma'] \in \mathbf{Sem}_{DEXP}$
- **cmdSem** : $cmd \rightarrow State \rightarrow Type * State$
 $(\forall c, \sigma) \text{ cmdSem}(c, \sigma) = (t, \sigma') \text{ iff } \langle c, \sigma \rangle \rightarrow (t, \sigma') \in \mathbf{Sem}_{CMD}$

● Stato: Strutture, Operazioni e Notazione

- (Δ, μ) : Stato con Stack di AR, Δ , e Store, μ .
- Costruttori: Tutte le strutture del Modello (Frame, AR, ...) hanno costruttori che forniscono Termini (Algebra libera) con cui esprimere tutte e sole le strutture di AM21.
- Operazioni: Tutte le strutture del Modello hanno operazioni di Accesso, Selezione, Modifica componenti per Pattern-Matching.

● Stack di AR

- $\gt ar_{top} \dots ar_1]$, struttura LIFO, con ar_{top} ultimo inserito e $top \geq 0$
- $\gt]$, Stack vuoto.
- $\#\Delta$, calcola size dello Stack, ie. numero di ar contenuti in esso.
- Pattern-Matching su termini Stack.

● Activation Record, AR.

- $\{\text{head}, \text{chain}, \text{frame}, \text{cont}, \text{val}\}$: AR a 5 componenti
- Head, intestazione: $\text{Ide} + \{[E], [C]\} + []$
- Chain, static chain: Offset espresso da intero $k \geq 0$
- Frame, di ambiente: Vedi sotto
- Cont, continuazione: Sequenza di comandi
- Val, return value: $\text{Val} + []$
- Pattern-Matching su termini AR.

● Frame: Operazioni.

- $[]$ Crea un frame vuoto per un AR.
- $[I/D] \otimes \Delta$ aggiunge il binding $[I/D]$ al frame di ar_{top} di Δ
- $\Delta(I)$ denotazione, D, del binding di I in Δ , se esiste ed in accordo allo scope di Small21.
- Pattern-Matching su termini frame

● Store: Operazioni

- $\triangleright(\mu, n)$ alloca in μ , n locazioni libere, in sequenza da loc, modifica μ in μ' , restituisce (loc, μ')
- $\triangleleft(\mu, \rho)$ dealloca da μ , le locazioni in ρ , che tornano allocabili. Restituisce la memoria modificata
- $\mu(\text{loc})$ (loc deve essere una locazione già allocata) restituisce il valore di loc
- $\mu[\text{loc} \leftarrow n]$ (loc deve essere già allocata) modifica il valore di loc con il valore n
- $\text{loc} \oplus k$ locazione k posizioni dopo loc.
- Pattern-Matching su termini store.

● Pattern-Matching

- Termine = Pattern: Lega le variabili del Pattern a termini quando può essere reso identico al Termine. Fallisce altrimenti.
- Uso. Posto in premessa di regole di inferenza, seleziona componenti del termine con cui instanziare la regola. In caso di fallimento rende la regola inapplicabile.

AM21: Il Modello Astratto - Le Operazioni 3

● Valori Calcolabili e Memorizzabili.

- $\text{INT} = \{0, 1, -1, \dots\}$
- $\text{BOOL} = \{\text{Cast}(\text{Bool}, 0), \text{Cast}(\text{Bool}, 1)\}$
- $\text{DVal} = \{\text{loc}_i^t \mid i \in [0..\text{StoreSize}], t \in \{\text{Int}, \text{Bool}, [\text{Arr}] t' k\}\}$

● Denotazioni.

- (t, v) per costanti: $t \in \text{Type}$, $v \in \{t\}$.
- $([\text{Mut}] t, \text{loc}_{t1})$ per modificabili: $[\text{Mut}] t \in \text{Type}$, loc_{t1} locazione per valori (memorizzabili) in $\{t\}$
Per i castable il valore utilizzato per la rappresentazione: Ad es. $([\text{Mut}] \text{Bool}, \text{loc}_{\text{Int}})$ per una variabile booleana quando i booleani sono memorizzabili ma non implementati (primitivi).
- $([\text{Mut}] ([\text{Arr}] t N), \text{loc}_{t1})$ per array a componenti modificabili: loc_{t1} locazione per valori nell'insieme utilizzato per la memorizzazione (vedi sopra).
- $\star I, t, (f_1, \dots, f_k), d, c, k$ closure per Procedure e Funzioni Procedurali con parametri.
 I = nome, t = tipo, (f_1, \dots, f_k) = formali, d = dichiarazioni, c = comando, k = posizione AR di dichiarazione nello Stack, della Procedura, Funzione Procedurale, dichiarata.

● Notazione: Simboli

- $[]$, Struttura Vuota o Valore di Default.
- \perp , Valore Indefinito.
- σ (anche con pedici, apici), Stato.
- Δ (anche con pedici, apici), Stack di AR.
- μ (anche con pedici, apici), Store.
- ρ (anche con pedici, apici), Frame.
- N (anche con pedici, apici), Intero.
- I (anche con pedici, apici), Identificatore.
- D , den (anche con pedici, apici), Denotazione.
- $\text{loc}, l, \text{loc}^t$ (anche con pedici), Locazione di memoria o word (di tipo t).
- ff , Sequenza, non vuota, parametri formali.
- aa , Sequenza, non vuota, parametri attuali.

Implementazione: Memoria, Frame e loro operazioni

Dobbiamo estendere AM21 con le strutture del Modello Astratto che definiscono l'Architettura della Macchina per Small21.

Lo facciamo partendo dai componenti base dello stato: Memoria e Frame (o Ambiente dei Locali)

Iniziamo QUI, con il fornire signature Ocaml alle operazioni previste sui valori Memoria e Frame

Continuiamo DOPO, con il fornire una rappresentazione adeguata per questi valori

● Memoria.

- **allocate**(μ, n): alloca n words (per interi) in sequenza
- **upd**(μ, loc, n): ($\text{alias}, \mu[\text{loc} \leftarrow n]$) modifica il valore di una locazione
- **getStore**(μ, loc): ($\text{alias}, \mu(\text{loc})$) fornisce il valore di una locazione
- **emptyStore**(): crea uno store iniziale con words libere, a valore indefinito

● Frame (o Ambiente Locale).

- **bind**($\rho, \text{ide}, \text{den}$): ($\text{alias}, [\text{ide}/\text{den}] \circ \rho$) aggiunge un nuovo binding
- **getEnv**(ρ, ide): ($\text{alias}, \rho(\text{ide})$) valore denotabile di un binding
- **emptyEnv**(): crea un'ambiente senza bindings

Implementazione: Rappresentazione della Memoria

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore Memoria e ogni valore Frame.

Useremo tipi algebrici

- **Memoria.**

```
type store = Store of int * (loc -> mval);;

(* Funzione di Astrazione ed Invariante di Rappresentazione per store:
1) Presentazione (o forma dei valori): [l1<-Mv1,...,lk<-Mvk] per k>=0
2) AF(c) = [] if c=Store(0,g)
   AF(c) = [l1<-Mv1,...,lk<-Mvk]
           if c=Store(k,g) and (perogni i\in[1..k], li=Loc(i-1) and (g li)=Mvi).
3) I(Store(n,g)) = 0<=n<=storeSize and
   (perogni i\in[1..n], li=Loc(i-1) and (g li) = Mval u and u\in [int]) and
   (perogni i\in[n+1..storeSize], li=Loc(i-1) and (g li) = Undef
*)
```

- **Frame.**

Implementazione: Rappresentazione del Frame

La rappresentazione stabilisce le strutture di dati con cui sarà rappresentato ogni valore Memoria e ogni valore Frame.

Useremo tipi algebrici

- Memoria.
- Frame.

```
type env = Env of ide list * (ide -> dval);;
```

```
(* Funzione di Astrazione ed Invariante di Rappresentazione per env:  
Modificata rispetto a SmallC: Includiamo anche il dominio ide list  
La presentazione non cambia. Cambiamenti nell'invariante e nelle  
operazioni (confronta definizione e loro uso)
```

```
1) Presentazione (o forma dei valori): [id1/D1,...,idk/Dk] per k>=0
```

```
2) AF(c) = [] if c = Env(l,g) and (perogni id, g(id) = Unbound)
```

```
AF(c) = [id1/D1,...,idk/Dk]
```

```
if c = Env(l,g) and (g(idi) = Di per i\in[1..k],
```

```
g(x) = Unbound altrimenti).
```

```
3) I(c) = (c = Env(l,g) => (List.mem x l) iff (g(x) != Unbound)
```

```
*)
```